Navigating the Code: A Qualitative Study of Novice Programmers' Perceptions and Utilization of Automated Feedback for Self-Regulated Learning

KELI LUO, University at Buffalo

This study investigates how novice programming students perceive and utilize automated feedback to support their self-regulated learning and metacognitive processes. Using a qualitative case study approach, we collected data from 20 students in an introductory programming course through interviews, think-aloud sessions, and reflective journals. Our findings reveal diverse perceptions of automated feedback, ranging from helpful tool to frustrating obstacle. We identified five main utilization strategies: immediate error correction, systematic debugging, concept reinforcement, progress monitoring, and ignore and continue. The study also found that automated feedback enhanced self-monitoring, promoted strategic planning, and triggered reflective thinking for many students, though its impact varied. These results contribute to our understanding of how automated feedback systems can be designed and implemented to better support students' self-regulated learning in programming education.

Keywords: automated feedback, self-regulated learning, metacognition, novice programmers

Reference Format:
Keli Luo. 2024. Navigating the Code: A Qualitative Study of Novice Programmers' Perceptions and Utilization of Automated Feedback for Self-Regulated Learning. *Education and Technology*, Vol. 1, Article 2 (September 2024), 10 pages.

Correspondence: Keli Luo, keliluo@buffalo.edu, University at Buffalo.

1 INTRODUCTION

In recent years, automated feedback systems have become increasingly prevalent in programming education, offering students immediate guidance on their code submissions. These systems promise to enhance the learning experience by providing timely, consistent feedback without increasing instructor workload [10]. However, the effectiveness of such systems depends not only on the quality of the feedback provided but also on how students perceive and utilize this feedback to support their learning processes.

Understanding how novice programming students engage with automated feedback is crucial for several reasons. First, it can inform the design of more effective feedback systems that align with students' needs and learning strategies. Second, it can help instructors better support students in developing self-regulated learning skills, which are essential for success in programming and beyond [27]. Finally, it can contribute to our broader understanding of how technology can scaffold metacognitive processes in complex learning domains.

Prior studies have made significant contributions to our understanding of automated feedback in programming education. For instance, Keuning et al. [11] conducted a systematic review of automated feedback generation systems, highlighting their potential benefits and limitations. Loksa and Ko [15] examined the role of self-regulation in programming problem-solving, emphasizing the importance of metacognitive strategies. However, these studies have primarily focused on the technical aspects of feedback systems or general self-regulation in programming, leaving a gap in our understanding of how students specifically interact with and utilize automated feedback to support their self-regulated learning and metacognitive processes.

To address this gap, our study investigates the research question: How do novice programming students perceive and utilize automated feedback to support their self-regulated learning and metacognitive processes while completing programming assignments? By examining this question, we aim to bridge the technical aspects of automated feedback systems with the psychological processes involved in learning to program.

Our study contributes to the existing literature in several ways. First, it provides a nuanced understanding of students' perceptions of automated feedback, moving beyond simple measures of satisfaction to explore how these perceptions influence learning behaviors. Second, it identifies specific strategies that students employ when engaging with automated feedback, offering insights into how these systems are used in practice. Third, it examines the impact of automated feedback on students' self-regulation and metacognitive processes, areas that have been underexplored in the context of programming education.

To investigate this question, we employed a qualitative case study approach, collecting rich data through interviews, think-aloud sessions, and reflective journals over the course of a semester-long introductory programming course. This methodology allowed us to capture the complexity of students' experiences and thought processes as they interacted with automated feedback [25].

Our findings reveal diverse perceptions and utilization strategies among novice programmers, as well as varying impacts on self-regulation and metacognition. These results have important implications for the design of automated feedback systems and pedagogical approaches in programming education. By deepening our understanding of how students engage with automated feedback, this study contributes to the ongoing efforts to improve programming education and support students in developing crucial self-regulated learning skills.

2 BACKGROUNDS

This literature review explores the intersections of self-regulated learning, metacognition, and automated feedback in the context of novice programming education. We examine the current

understanding of these concepts and their applications in computer science education, highlighting the gaps that our research aims to address.

2.1 Self-Regulated Learning in Programming Education

Self-regulated learning (SRL) has been recognized as a crucial factor in academic success across various disciplines, including computer science. In the context of programming education, SRL refers to the process by which students actively monitor and control their cognition, motivation, and behavior to achieve their learning goals [26].

Falkner et al. [6] identified a set of SRL strategies specific to computer science education, including task difficulty assessment, designing before coding, and problem decomposition. Their study highlighted the importance of explicit instruction in these strategies, particularly for novice programmers who may not naturally develop effective self-regulation skills.

Further research by Loksa and Ko [16] demonstrated that successful SRL in programming relies on adequate domain knowledge and that students who engage in more planning and comprehension monitoring tend to produce code with fewer errors. This underscores the need for interventions that not only teach programming concepts but also foster the development of SRL skills.

2.2 Metacognition in Novice Programmers

Metacognition, often considered a key component of SRL, involves the awareness and regulation of one's own cognitive processes. In programming education, metacognition plays a crucial role in problem-solving, debugging, and overall learning effectiveness [19].

Loksa et al. [18] investigated novice programmers' in-situ reflections on their programming process, revealing that many students struggle with basic metacognitive tasks such as assessing their own understanding and progress. This finding suggests a need for explicit support in developing metacognitive skills within programming environments.

Moreover, Arakawa et al. [1] identified specific self-regulated learning struggles that novice programmers face, including difficulties in task analysis, self-control, and self-reflection. Their study highlighted the importance of providing targeted support for these metacognitive processes, especially as students approach assignment deadlines.

2.3 Automated Feedback in Programming Courses

Automated feedback systems have become increasingly prevalent in programming education, offering scalable solutions for providing timely and consistent feedback to large numbers of students [8]. These systems typically assess student code submissions against predefined test cases and provide immediate feedback on correctness and potential errors.

Research by Keuning et al. [12] reviewed various automated feedback generation systems for programming exercises, categorizing the types of feedback provided and their effectiveness. They found that while many systems focus on identifying errors, fewer offer guidance on how to proceed or support higher-level learning strategies. This gap highlights the need for more sophisticated feedback mechanisms that not only identify issues but also scaffold the learning process.

A study by Qiang et al. [21] explored how the delivery of automated feedback affects knowledge transfer in programming education. Their findings suggest that the format and timing of feedback can significantly impact its effectiveness, highlighting the need for careful design of automated feedback systems to maximize learning outcomes. Building on this, Becker et al. [4] investigated the landscape of compiler error messages, emphasizing the importance of clear and comprehensible feedback for novice programmers.

Recent work by Wu et al. [23] examined the effects of automated feedback on students' self-regulated learning in introductory programming courses. They found that while automated feedback

can support SRL processes, its effectiveness varies depending on students' prior programming experience and metacognitive skills. This underscores the need for adaptive feedback systems that can cater to individual student needs. Furthermore, Hundhausen et al. [9] explored the use of social learning analytics to enhance automated feedback in programming courses. Their approach combined automated assessment with peer code review, demonstrating the potential for integrating social learning aspects into automated feedback systems.

2.4 Integration of SRL, Metacognition, and Automated Feedback

While separate bodies of research exist on SRL, metacognition, and automated feedback in programming education, fewer studies have examined the intersection of these areas, particularly from the perspective of novice programmers.

Silva et al. [22] developed a computer-supported collaborative learning environment that incorporated scaffolding for both individual and social regulation of learning in programming education. Their study demonstrated the potential for integrating SRL support into programming environments but did not specifically focus on the role of automated feedback in this process. Similarly, Prather et al. [20] investigated the use of metacognitive scaffolding in interpreting programming problem prompts, showing positive effects on student performance. However, their study did not explore how students utilize automated feedback as part of their metacognitive processes.

Recent work by Li et al. [13] examined the influence of socially shared regulation on computational thinking performance in cooperative learning environments. While their study focused on collaborative aspects, it highlighted the potential for integrating regulatory support into programming tools, including automated feedback systems.

Xie et al. [24] proposed a theory of instruction for introductory programming skills that emphasizes the importance of metacognitive support. Their work suggests that automated feedback could play a crucial role in fostering metacognitive awareness and self-regulation in novice programmers. In a related vein, Grey and Gordon [7] explored the use of gamified programming tutors to motivate students in learning to write code. Their approach incorporated elements of automated feedback and self-regulation, demonstrating the potential for gamification to enhance SRL in programming education. Despite these advancements, there remains a significant gap in understanding how novice programmers perceive and utilize automated feedback in relation to their self-regulated learning and metacognitive processes. Loksa et al. [17] reviewed the state of metacognition and self-regulation research in programming education, highlighting the need for more integrated approaches that combine automated tools with explicit support for regulatory skills. This gap in the literature underscores the importance of our current study, which aims to provide a deeper understanding of how automated feedback can be leveraged to support SRL and metacognition in novice programmers. By exploring students' perceptions and utilization of automated feedback, we can inform the design of more effective learning environments that foster both programming skills and self-regulatory abilities.

3 RESEARCH DESIGN

This study aimed to address the following research question: How do novice programming students perceive and utilize automated feedback to support their self-regulated learning and metacognitive processes while completing programming assignments?

To investigate this question, we employed a qualitative case study approach. Case studies allow for in-depth exploration of a phenomenon within its real-world context [25]. This design was appropriate given our focus on understanding students' experiences and thought processes related to automated feedback in an authentic programming course setting.

3.1 Context and Participants

The study took place in an introductory programming course (CS1) at a large public university in the northeastern United States. The course taught Python programming and enrolled approximately 200 students, primarily first-year computer science and engineering majors. We recruited 20 students to participate in the study. Participants were selected using purposeful sampling to ensure a range of prior programming experience and academic performance levels were represented. The final sample included 12 male and 8 female students.

3.2 Data Collection

Data was collected over the course of one 15-week semester using multiple qualitative methods.

3.2.1 Semi-structured Interviews. We conducted three 45-60 minute semi-structured interviews with each participant - at the beginning, middle, and end of the semester. Interview protocols focused on students' approaches to programming assignments, their use of automated feedback, and their metacognitive and self-regulatory processes. Sample questions included:

- How do you typically approach programming assignments in this course?
- What do you do when you receive automated feedback on your code?
- How, if at all, does the automated feedback influence how you think about and regulate your learning process?

3.2.2 Think-aloud Sessions. Participants completed two think-aloud sessions while working on course programming assignments. These 60-90 minute sessions took place in weeks 5 and 10 of the semester. Students were asked to verbalize their thoughts and actions as they coded and interacted with the automated feedback system. The researcher observed and took detailed field notes.

3.2.3 Reflective Journal Entries. Students completed weekly reflective journal entries responding to prompts about their programming experiences and use of automated feedback. Prompts included:

- What challenges did you face in your programming assignment this week?
- How did you use the automated feedback? Was it helpful? Why or why not?
- What strategies did you use to monitor and regulate your learning process?

3.3 Data Analysis

Data analysis followed an iterative, inductive approach guided by thematic analysis techniques [5]. The analysis process involved the following steps:

(1) Transcription of all interview and think-aloud recordings
(2) Initial open coding of a subset of data to develop preliminary codes
(3) Development of a codebook with code definitions and examples
(4) Coding of full dataset by two independent coders
(5) Calculation of inter-rater reliability (Cohen's kappa)
(6) Discussion and consensus on coding discrepancies
(7) Identification of key themes and patterns across the dataset
(8) Development of thematic map to illustrate relationships between themes
(9) Selection of illustrative quotes and examples for each theme

To enhance trustworthiness, we employed several strategies including triangulation of data sources, member checking with participants, peer debriefing, and maintaining an audit trail of analysis decisions [14]. The qualitative analysis software NVivo 12 was used to assist with data

management and coding. Coding focused on identifying students' perceptions of automated feedback, their reported and observed uses of feedback, and their metacognitive and self-regulatory processes.

4 RESULTS

Our analysis of interview transcripts, think-aloud session notes, and reflective journal entries revealed several key themes related to how novice programming students perceive and utilize automated feedback to support their self-regulated learning and metacognitive processes. We present these findings organized around three main themes: (1) perceptions of automated feedback, (2) utilization strategies, and (3) impact on self-regulation and metacognition.

4.1 Perceptions of Automated Feedback

Students' perceptions of the automated feedback system varied, but generally fell into three categories: helpful tool, frustrating obstacle, or neutral resource. The majority of participants (15 out of 20) viewed the automated feedback as a helpful tool that supported their learning process. These students appreciated the immediate nature of the feedback and saw it as a way to efficiently identify and correct errors. For example, one student noted:

> "The automated feedback is like having a tutor available 24/7. It helps me catch silly mistakes quickly and gives me confidence that I'm on the right track." (P7, Interview 2)

A minority of students (3 out of 20) perceived the automated feedback as a frustrating obstacle to their programming process. These participants often felt that the feedback was too vague or unhelpful. As one student explained:

> "Sometimes the feedback is just cryptic error messages. It doesn't actually help me understand what's wrong or how to fix it. It's just frustrating." (P12, Interview 3)

The remaining students (2 out of 20) had neutral perceptions of the automated feedback, viewing it as just another available resource. These students neither strongly appreciated nor disliked the feedback system.

4.2 Utilization Strategies

Analysis of think-aloud sessions and reflective journals revealed five primary strategies students employed when utilizing the automated feedback:

(1) Immediate error correction
(2) Systematic debugging
(3) Concept reinforcement
(4) Progress monitoring
(5) Ignore and continue

Table 1 provides an overview of these strategies, including their frequency of use and representative quotes.

4.3 Impact on Self-Regulation and Metacognition

Our analysis revealed that the automated feedback system influenced students' self-regulated learning and metacognitive processes in several ways.

Many students (16 out of 20) reported that the automated feedback enhanced their ability to monitor their own learning progress. The immediate nature of the feedback allowed them to quickly identify areas where they needed improvement. As one student explained:

Strategy	Frequency	Representative Quote
Immediate error correction	85%	"As soon as I see an error in the feedback, I try to fix it right away." (P3, Think-aloud 1)
Systematic debugging	60%	"I use the feedback to guide my debugging process. I go through each error methodically." (P15, Journal Week 7)
Concept reinforcement	45%	"The feedback helps me understand concepts better. When I get an error, I go back to my notes to review." (P9, Interview 2)
Progress monitoring	40%	"I use the feedback to track my progress. It's satisfying to see the number of errors decrease." (P18, Think-aloud 2)
Ignore and continue	15%	"Sometimes I just ignore the feedback and keep coding. I'll deal with errors later." (P6, Interview 3)

Table 1. Automated Feedback Utilization Strategies

> "The feedback helps me keep track of what I understand and what I don't. It's like a constant check on my learning." (P5, Interview 3)

Some students (12 out of 20) noted that interacting with the automated feedback system encouraged them to engage in more strategic planning. These students would use the feedback to inform their approach to future programming tasks. For example:

> "After getting feedback on one assignment, I started planning out my code more carefully for the next one. I tried to anticipate the kinds of errors the system might catch." (P14, Journal Week 10)

For about half of the participants (11 out of 20), the automated feedback triggered more reflective thinking about their programming process. These students would often pause to consider why they received certain feedback and how it related to their understanding of programming concepts. One student described this process:

> "When I get unexpected feedback, it makes me stop and think. Why did I make that mistake? What was I thinking? It helps me understand my own thought process better." (P2, Think-aloud 2)

It's important to note that for a subset of students (4 out of 20), the automated feedback appeared to have limited impact on their self-regulation and metacognitive processes. These students tended to use the feedback in a more superficial way, focusing solely on correcting errors without deeper reflection.

> "I just use the feedback to fix errors. I don't really think about it beyond that." (P20, Interview 3)

Overall, our findings suggest that while the majority of students found the automated feedback system beneficial for supporting their self-regulated learning and metacognitive processes, the impact varied based on individual perceptions and utilization strategies.

5 DISCUSSION

This study investigated how novice programming students perceive and utilize automated feedback to support their self-regulated learning and metacognitive processes. Our findings provide insights into students' experiences with automated feedback and its impact on their learning strategies. In this section, we discuss the implications of our results, connect them to existing literature, and consider limitations and future directions.

5.1 Perceptions and Utilization of Automated Feedback

Our results revealed diverse perceptions of automated feedback among novice programmers, with the majority viewing it as a helpful tool. This aligns with previous research by Keuning et al. [11], who found that students generally appreciate automated feedback in programming courses. However, our study extends this understanding by identifying a subset of students who perceive automated feedback as a frustrating obstacle or neutral resource.

The utilization strategies we identified (immediate error correction, systematic debugging, concept reinforcement, progress monitoring, and ignore and continue) demonstrate the varied ways students engage with automated feedback. These findings support and expand upon the work of Loksa and Ko [16], who emphasized the importance of problem-solving strategies in programming education. Our results suggest that automated feedback can scaffold these problem-solving processes, but students' approaches to using the feedback differ.

Interestingly, we found that some students used automated feedback for concept reinforcement, a strategy not prominently discussed in previous literature on automated assessment in programming education [10]. This suggests that automated feedback systems may have untapped potential for supporting conceptual understanding, not just error correction.

5.2 Impact on Self-Regulation and Metacognition

Our findings indicate that automated feedback can positively influence students' self-regulated learning and metacognitive processes, particularly in enhancing self-monitoring and promoting strategic planning. This aligns with the theoretical framework proposed by Zimmerman [27], which emphasizes the cyclical nature of self-regulated learning.

The enhancement of self-monitoring through automated feedback is particularly noteworthy. As Azevedo [3] argued, effective self-monitoring is crucial for successful learning in complex domains like computer programming. Our results suggest that automated feedback systems can serve as external monitors, potentially scaffolding students' development of internal monitoring skills. However, our observation that some students engaged in more reflective thinking due to automated feedback while others used it more superficially highlights the variable impact of such systems. This variability echoes the findings of Arakawa et al. [2], who identified different levels of metacognitive awareness among programming students. Our study extends this work by linking these differences specifically to interactions with automated feedback.

5.3 Limitations and Future Directions

While this study provides valuable insights, it has several limitations. First, our sample size was relatively small and drawn from a single institution, which limits generalizability. Future research could expand to multiple institutions and larger sample sizes to validate and extend our findings.

Second, our study focused on novice programmers in an introductory course. It would be valuable to investigate how perceptions and utilization of automated feedback change as students progress in their programming education. Longitudinal studies could provide insights into this developmental process.

Third, while our qualitative approach provided rich data on students' experiences, it did not allow us to quantitatively measure the impact of automated feedback on learning outcomes. Future studies could employ mixed methods approaches to correlate students' perceptions and utilization of feedback with their performance in programming tasks. Finally, our study did not explore the specific features of the automated feedback system that influenced students' perceptions and utilization. Future research could investigate how different types of feedback (e.g., more or less detailed, focusing on different aspects of code) impact students' engagement and learning processes.

In conclusion, this study contributes to our understanding of how novice programmers engage with automated feedback and how it impacts their self-regulated learning and metacognitive processes. These findings can inform the design of more effective automated feedback systems and pedagogical approaches in programming education.

6 CONCLUSION

This study provides valuable insights into how novice programming students engage with automated feedback and its impact on their self-regulated learning and metacognitive processes. Our findings highlight the diverse ways students perceive and utilize automated feedback, as well as its potential to enhance self-monitoring, strategic planning, and reflective thinking. However, the variable impact across students underscores the need for tailored approaches in implementing these systems. Future research should explore how perceptions and utilization strategies evolve as students progress in their programming education, and investigate the specific features of automated feedback systems that most effectively support self-regulated learning. By deepening our understanding of these processes, this study contributes to the ongoing efforts to improve programming education and support students in developing crucial self-regulated learning skills. Ultimately, these insights can inform the design of more effective automated feedback systems and pedagogical approaches, potentially improving learning outcomes and retention in programming courses.

REFERENCES

[1] Kai Arakawa, Qiang Hao, Tyler Greer, Lu Ding, Christopher D. Hundhausen, and Abigayle Peterson. 2021. In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. ACM, 467–473. https://doi.org/10.1145/3408877.3432357

[2] Kai Arakawa, Qiang Hao, Tyler Greer, Lu Ding, Christopher D Hundhausen, and Abigayle Peterson. 2021. In Situ Identification of Student Self-Regulated Learning Struggles in Programming Assignments. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. 467–473.

[3] Roger Azevedo. 2009. Theoretical, conceptual, methodological, and instructional issues in research on metacognition and self-regulated learning: A discussion. *Metacognition and Learning* 4, 1 (2009), 87–95.

[4] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (2019), 177–210. https://doi.org/10.1145/3344429.3372508

[5] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[6] Katrina Falkner, Rebecca Vivian, and Nickolas J.G. Falkner. 2014. Identifying Computer Science Self-Regulated Learning Strategies. In *Proceedings of the 2014 Conference on Innovation Technology in Computer Science Education*. ACM, 291–296. https://doi.org/10.1145/2591708.2591715

[7] Simon Grey and Neil A. Gordon. 2023. Motivating Students to Learn How to Write Code Using a Gamified Programming Tutor. *Education Sciences* 13, 3 (2023), 230. https://doi.org/10.3390/educsci13030230

[8] Qiang Hao and Michail Tsikerdekis. 2019. How Automated Feedback Is Delivered Matters: Formative Feedback and Knowledge Transfer. In *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–6. https://doi.org/10.1109/FIE43999.2019.9028444

[9] Christopher D. Hundhausen, Anukrati Agrawal, and Pawan Agarwal. 2017. Talking about Code: Integrating Pedagogical Code Reviews into Early Computing Courses. In *Proceedings of the 2013 ACM SIGCSE Technical Symposium on Computer*

Science Education. ACM, 541–546. https://doi.org/10.1145/2445196.2445346

[10] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Sepp"al"a. 2010. Review of recent systems for automatic assessment of programming assignments. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (2010), 86–93.

[11] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43.

[12] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* 19, 1 (2018), 1–43. https://doi.org/10.1145/3231711

[13] Juan Li, Jingjing Liu, Ruiying Yuan, and Rustam Shadiev. 2022. The Influence of Socially Shared Regulation on Computational Thinking Performance in Cooperative Learning. *Educational Technology Society* 25, 1 (2022), 48–60. https://doi.org/10.30191/ETS.202201251.0004

[14] Yvonna S Lincoln and Egon G Guba. 1985. *Naturalistic inquiry*. Sage.

[15] Dastyni Loksa and Andrew J Ko. 2016. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 83–91.

[16] Dastyni Loksa and Andrew J. Ko. 2016. The Role of Self-Regulation in Programming Problem Solving Process and Success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM, 83–91. https://doi.org/10.1145/2960310.2960334

[17] Dastyni Loksa, Lauren Margulieux, Brett A. Becker, Michelle Craig, Paul Denny, Raymond Pettit, and James Prather. 2022. Metacognition and Self-Regulation in Programming Education: Theories and Exemplars of Use. *ACM Transactions on Computing Education* 22, 4 (2022), 1–31. https://doi.org/10.1145/3487050

[18] Dastyni Loksa, Benjamin Xie, Harrison Kwik, and Amy J. Ko. 2020. Investigating Novices' In Situ Reflections on Their Programming Process. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, 149–155. https://doi.org/10.1145/3328778.3366846

[19] James Prather, Brett A. Becker, Michelle Craig, Paul Denny, Dastyni Loksa, and Lauren Margulieux. 2020. What Do We Think We Think We Are Doing? Metacognition and Self-Regulation in Programming. *Proceedings of the 2020 ACM Conference on International Computing Education Research* (2020), 2–13. https://doi.org/10.1145/3372782.3406263

[20] James Prather, Raymond Pettit, Brett A. Becker, Paul Denny, Dastyni Loksa, Alani Peters, Zachary Albrecht, and Krista Masci. 2019. First Things First: Providing Metacognitive Scaffolding for Interpreting Problem Prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, 531–537. https://doi.org/10.1145/3287324.3287374

[21] Hao Qiang, Jack Wilson, Camille Ottaway, Naitra Iriumi, Kai Arakawa, and David H. Smith. 2019. Investigating the Essential of Meaningful Automated Formative Feedback for Programming Assignments. In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 151–155. https://doi.org/10.1109/VLHCC.2019.8818815

[22] Leonardo Silva, António Mendes, Anabela Gomes, and Gabriel Fortes. 2023. Fostering Regulation of Learning Processes Among Programming Students Using Computational Scaffolding. *International Journal of Computer-Supported Collaborative Learning* 18, 1 (2023), 67–100. https://doi.org/10.1007/s11412-023-09388-y

[23] Yifei Wu, Xiaofeng Wei, Meishan Liu, and Yizhou Qian. 2024. Exploring the Effects of Automated Feedback on Students in Introductory Programming Using Self-regulated Learning Theory. *ACM Turing Award Celebration Conference 2024* (2024), 76–80. https://doi.org/10.1145/3674399.3674430

[24] Benjamin Xie, Dastyni Loksa, Greg L. Nelson, Matthew J. Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J. Ko. 2019. A Theory of Instruction for Introductory Programming Skills. *Computer Science Education* 29, 2-3 (2019), 205–253. https://doi.org/10.1080/08993408.2019.1565235

[25] Robert K Yin. 2017. *Case study research and applications: Design and methods*. Sage publications.

[26] Barry J. Zimmerman. 2000. Attaining Self-Regulation: A Social Cognitive Perspective. *Handbook of Self-Regulation* (2000), 13–39. https://doi.org/10.1016/B978-012109890-2/50031-7

[27] Barry J Zimmerman. 2002. Becoming a self-regulated learner: An overview. *Theory into practice* 41, 2 (2002), 64–70.